Self Healing Infrastructure - Project Report

Introduction Modern IT systems require high availability and resilience. Manual intervention for service recovery can lead to increased downtime and operational costs. This project demonstrates a self-healing infrastructure that automatically detects and recovers from service failures using open-source tools. The goal is to minimize downtime and ensure continuous service availability by automating the detection and remediation of failures.

Background

Traditional infrastructure monitoring solutions can detect failures but often require manual intervention to resolve issues. This can result in prolonged outages and increased operational burden. Self-healing systems address this by integrating monitoring, alerting, and automated remediation, enabling systems to recover from failures without human intervention. This project leverages widely adopted open-source tools to build such a system in a modular and extensible way.

Abstract The Self Healing Infrastructure project integrates Prometheus for monitoring, Alertmanager for alerting, and Ansible for automated recovery. When a monitored service (e.g., Nginx) fails, Prometheus detects the issue and triggers an alert. Alertmanager forwards the alert to a custom webhook application, which then executes an Ansible playbook to restore the service. All components are containerized using Docker Compose for easy deployment and management.

The system is designed to be extensible, allowing additional services and recovery playbooks to be added with minimal configuration changes. The use of containers ensures portability and simplifies the setup process, making it suitable for both development and production environments.

Tools Used

- **Prometheus:** For monitoring service health and metrics collection.
- **Alertmanager:** For managing and routing alerts from Prometheus.
- **Ansible:** For automating recovery actions via playbooks.
- **Docker & Docker Compose:** For containerizing and orchestrating all components.
- **Flask (Python):** For the webhook application that receives alerts and triggers Ansible.

Architecture Overview

The architecture consists of the following components:

- **Prometheus:** Collects metrics from monitored services and applies alerting rules defined in `alert.rules.yml`.
- **Alertmanager:** Receives alerts from Prometheus and routes them to the configured webhook endpoint.

- **Webhook Application (Flask):** Listens for incoming alerts and triggers the appropriate Ansible playbook based on the alert content.
- **Ansible Playbooks:** Automate the recovery process, such as restarting a failed Nginx service.
- **Docker Compose:** Orchestrates the deployment of all components, ensuring they run in isolated containers.

Steps Involved in Building the Project

1. **Set up Prometheus:**
   - Configure `prometheus.yml` to scrape metrics and apply alerting rules from `alert.rules.yml`.
2. **Configure Alertmanager:**
   - Set up `alertmanager.yml` to send alerts to the webhook endpoint.
3. **Develop the Webhook Application:**
   - Create `webhook_app.py` using Flask to receive alerts and execute Ansible playbooks.
4. **Write Ansible Playbooks:**
   - Implement `playbooks/recover_nginx.yml` to restart the Nginx service upon failure.
5. **Containerize Components:**
   - Write `Dockerfile` and `docker-compose.yml` to build and run all services together.
6. **Testing:**
   - Simulate a service failure and verify that the system detects, alerts, and recovers automatically.

Configuration Details

- **Prometheus Configuration:**
  - The `prometheus.yml` file specifies scrape targets and includes the alerting rules from `alert.rules.yml`. These rules define the conditions under which alerts are fired (e.g., Nginx service down).
- **Alertmanager Configuration:**
  - The `alertmanager.yml` file configures the receiver for alerts, specifying the webhook endpoint provided by the Flask app.
- **Webhook Application:**
  - The Flask app (`webhook_app.py`) parses incoming alert payloads and determines which Ansible playbook to execute. It uses Python's `subprocess` module to run Ansible commands.
- **Ansible Playbooks:**
  - The playbook `recover_nginx.yml` contains tasks to restart the Nginx service. Additional playbooks can be added for other services as needed.
- **Docker Compose:**
  - The `docker-compose.yml` file defines services for Prometheus, Alertmanager, and the webhook app, ensuring they are networked together.

Testing Methodology

To validate the self-healing capability, the following test procedure was used:

1. Start all services using Docker Compose.
2. Manually stop the Nginx service to simulate a failure.
3. Observe Prometheus detecting the failure and firing an alert.
4. Confirm that Alertmanager sends the alert to the webhook app.
5. Verify that the webhook app triggers the Ansible playbook, which restarts Nginx.
6. Check Prometheus to ensure the service is restored and the alert is resolved.

Results

The system successfully detected and recovered from simulated Nginx failures with minimal delay. Alerts were generated and resolved automatically, demonstrating the effectiveness of the self-healing approach. The modular design allows for easy extension to other services and recovery scenarios.

Snapshots and Descriptions

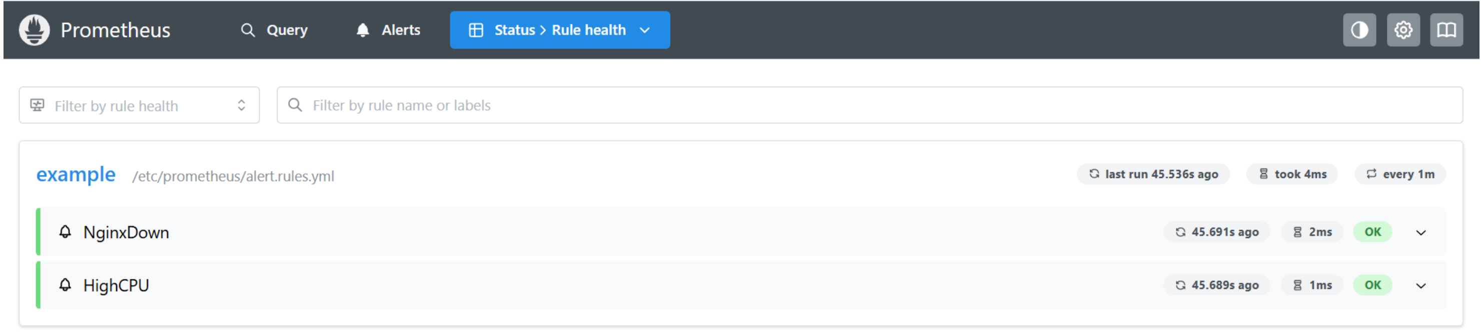
The following snapshots illustrate the monitoring and alerting workflow of the self-healing infrastructure:

1. Prometheus Alerts View

<img width="1919" height="796" alt="Screenshot 2025-08-19 201617" src="https://github.com/user-attachments/assets/85e68c36-4a71-4c51-8f35-b0431285e216" />

Description: This snapshot shows the Prometheus Alerts page, where two alert rules are configured: `NginxDown` and `HighCPU`. The `NginxDown` alert is in a pending state, indicating that Prometheus has detected a potential issue with the Nginx service. The `HighCPU` alert is also present, demonstrating the system's ability to monitor multiple conditions. This view provides real-time visibility into the health of monitored services and the status of alert rules.

2. Prometheus Rule Health



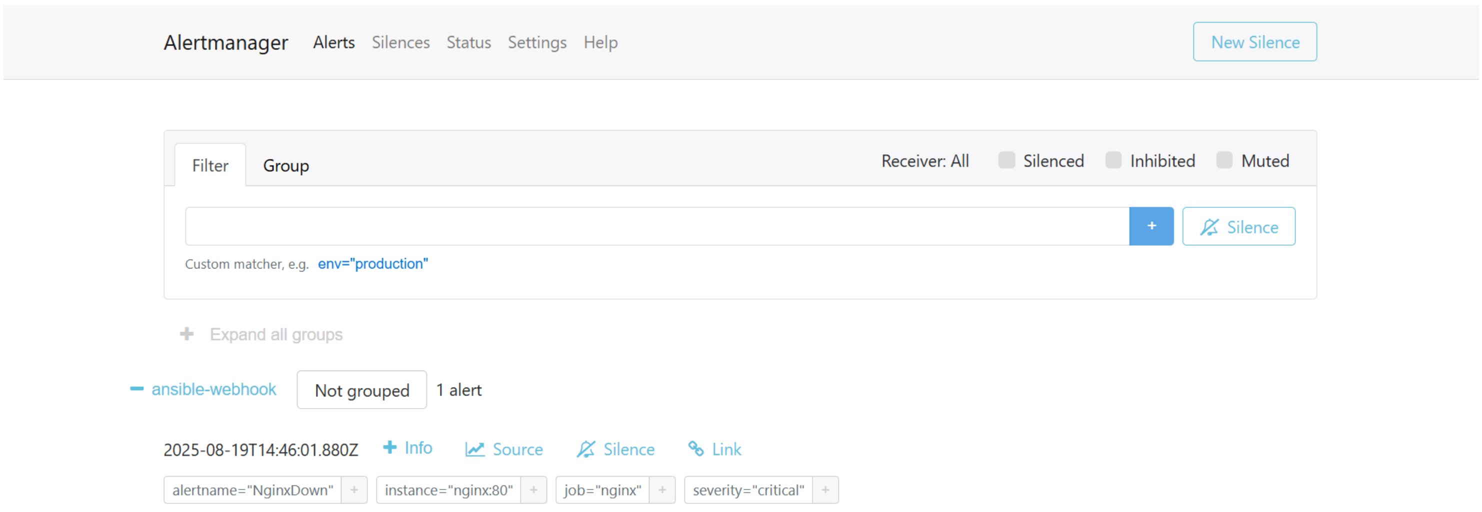
example /etc/prometheus/alert.rules.yml last run 45.536s ago took 4ms every 1m

NginxDown 45.691s ago 2ms OK

HighCPU 45.689s ago 1ms OK

Description: This snapshot displays the health status of alerting rules in Prometheus. Both `NginxDown` and `HighCPU` rules are shown as healthy (OK), indicating that Prometheus is actively evaluating these rules and has not detected any current issues. The last run time and evaluation duration are also visible, confirming that the monitoring system is functioning as expected.

3. Alertmanager Alert Details



ansible-webhook Not grouped 1 alert

2025-08-19T14:46:01.880Z Info Source Silence Link

alertname="NginxDown" instance="nginx:80" job="nginx" severity="critical"

Description: This snapshot captures the Alertmanager interface, where an active alert (`NginxDown`) has been received and routed to the `ansible-webhook` receiver. The alert details include the alert name, instance, job, and severity. This view demonstrates how Alertmanager processes and displays incoming alerts, and how it can be configured to forward alerts to external systems (such as the webhook application for automated remediation).

Conclusion This project demonstrates a practical approach to building resilient, self-healing infrastructure using open-source tools. By automating failure detection and recovery, organizations can reduce downtime and improve service reliability. The modular, containerized design ensures easy deployment and scalability for real-world applications.

Future Work

- Integrate additional monitoring and alerting for other critical services.
- Implement more sophisticated remediation strategies (e.g., rolling restarts, scaling).
- Add a dashboard for real-time visualization of alerts and recovery actions.
- Explore integration with cloud-native solutions (e.g., Kubernetes, cloud monitoring APIs).